Week 4: NLP Disaster Tweets Kaggle Mini-Project

Introduction:

In today's digital age, Twitter has emerged as a crucial communication platform, particularly during times of emergency. The prevalence of smartphones allows individuals to promptly share real-time observations of unfolding emergencies. Consequently, there is a growing interest among various agencies, including disaster relief organizations and news agencies, to programmatically monitor Twitter for timely and relevant information.

However, distinguishing whether a person's words on Twitter truly indicate a disaster poses a unique challenge. Consider the following example:

The author explicitly uses the word “ABLAZE” but means it metaphorically. While this metaphorical usage may be immediately apparent to a human, especially with the aid of visual context, it becomes less clear to a machine.

In the context of this competition, participants are tasked with developing a machine learning model capable of discerning which tweets pertain to real disasters and which ones do not. The dataset for this project comprises 10,000 tweets that have been manually classified. Whether you are a seasoned NLP practitioner or a newcomer to the field, we have prepared a brief tutorial to help you swiftly familiarize yourself with the essentials of Natural Language Processing (NLP) and get started on this exciting challenge.

Data Description:

Each sample in the train and test set has the following information:

- The text of a tweet
- A keyword from that tweet (although this may be blank!)
- The location the tweet was sent from (may also be blank)

Dataset:

- The project outlien and dataset are available from Kaggle.
- Link to the Kaggle project site:https://www.kaggle.com/competitions/nlp-getting-started/data (https://www.kaggle.com/competitions/nlp-getting-started/data)

Objective:

- Predicting whether a given tweet is about a real disaster or not. If so, predict a 1. If not, predict a 0.

Acknowledgments:

This dataset was created by the company figure-eight and originally shared on their ‘Data For Everyone’ website here.

Tweet source: https://twitter.com/AnyOtherAnnaK/status/629195955506708480 (https://twitter.com/AnyOtherAnnaK/status/629195955506708480)

Importing the necessary libraries

In [175]:

```
#Importing Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
import tensorflow as tf

import keras
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, RNN, LSTM, GRU, Dropout, Dense
from keras import layers
import keras_tuner

import nltk
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize, sent_tokenize
import spacy

import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os

from wordcloud import WordCloud, STOPWORDS
from bs4 import BeautifulSoup
import contractions
import re, string, unicodedata

from sklearn.feature_extraction.text import CountVectorizer # Import count Vectorizer
from sklearn.model_selection import train_test_split # Import train test split
from sklearn.ensemble import RandomForestClassifier # Import Rndom Forest Classifier
from sklearn.model_selection import cross_val_score # Import cross val score
from sklearn.metrics import confusion_matrix # Import confusion matrix
from sklearn.feature_extraction.text import TfidfVectorizer # Import Tf-Idf vector
```

Reading the dataset

In [188]:

```
tweets_train = pd.read_csv('train.csv')
tweets_test = pd.read_csv('test.csv')
```

Initial Exploration of the Dataset

In [189]:
```
tweets_train.shape
```

Out[189]:
```
(7613, 5)
```

In [190]:
```
tweets_train.head()
```

Out[190]:

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

In [191]:
```
tweets_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        7613 non-null   int64 
 1   keyword   7552 non-null   object
 2   location  5080 non-null   object
 3   text      7613 non-null   object
 4   target    7613 non-null   int64 
dtypes: int64(2), object(3)
memory usage: 297.5+ KB
```

In [192]:
```
tweets_test.shape
```

Out[192]:
```
(3263, 4)
```

In [193]:
```
tweets_test.head()
```

Out[193]:

	id	keyword	location	text
0	0	NaN	NaN	Just happened a terrible car crash
1	2	NaN	NaN	Heard about #earthquake is different cities, s...
2	3	NaN	NaN	there is a forest fire at spot pond, geese are...
3	9	NaN	NaN	Apocalypse lighting. #Spokane #wildfires
4	11	NaN	NaN	Typhoon Soudelor kills 28 in China and Taiwan

In [194]:
```
tweets_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        3263 non-null   int64 
 1   keyword   3237 non-null   object
 2   location  2158 non-null   object
 3   text      3263 non-null   object
dtypes: int64(1), object(3)
memory usage: 102.1+ KB
```

The training dataset comprises a total of 7613 records and is structured with five columns: id, keyword, location, text, and target. Among these, 'id' and 'target' are numeric, while the remaining three features are of string type. Notably, the 'keyword' and 'location' columns have some null values. For the purpose of this project, both columns are slated for removal as they will not be utilized. Similar operations are applied to the testing data to maintain consistency in data preprocessing.

Initial Data Preprocessing

In [195]:
```
tweets_train.isnull().sum(axis=0)
```

Out[195]:
```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

In [196]:
```
tweets_test.isnull().sum(axis=0)
```

Out[196]:
```
id             0
keyword       26
location    1105
text           0
dtype: int64
```

There seems to be a large percentage of 'location' features missing. It is probably prudent that we drop this variable. 'Keyword' is also missing some features, but it is a small percentage. We may be able to infer the keyword, or drop the missing rows.

In [197]:
```
train_clean = tweets_train.drop(['keyword', 'location'], axis = 1)
train_clean.head()
```

Out[197]:

	id	text	target
0	1	Our Deeds are the Reason of this #earthquake M...	1
1	4	Forest fire near La Ronge Sask. Canada	1
2	5	All residents asked to 'shelter in place' are ...	1
3	6	13,000 people receive #wildfires evacuation or...	1
4	7	Just got sent this photo from Ruby #Alaska as ...	1

In [198]:
```
test_clean = tweets_test.drop(['keyword', 'location'], axis = 1)
test_clean.head()
```

Out[198]:

	id	text
0	0	Just happened a terrible car crash
1	2	Heard about #earthquake is different cities, s...
2	3	there is a forest fire at spot pond, geese are...
3	9	Apocalypse lighting. #Spokane #wildfires
4	11	Typhoon Soudelor kills 28 in China and Taiwan

In [199]:
```
# check duplicates
train_clean[train_clean.duplicated()]
```

Out[199]:

id	text	target

In [200]:
```
train_clean[train_clean.duplicated()]
```

Out[200]:

id	text	target

In [201]:
```
tweets_full = train_clean.assign(dataset_ind = np.repeat('train', train_clean.shape[0])).a
print(tweets_full[tweets_full.duplicated(keep = False)].to_string())
```

```
Empty DataFrame
Columns: [id, text, dataset_ind]
Index: []
```

Both training and test set have null values dropped. In addition, training set has 92 duplicated records and duplicates have been dropped since they don't add any value to the future models. Besides, training and test set don't share tweets in common. That means the model built in later steps won't suffered from data leakage.

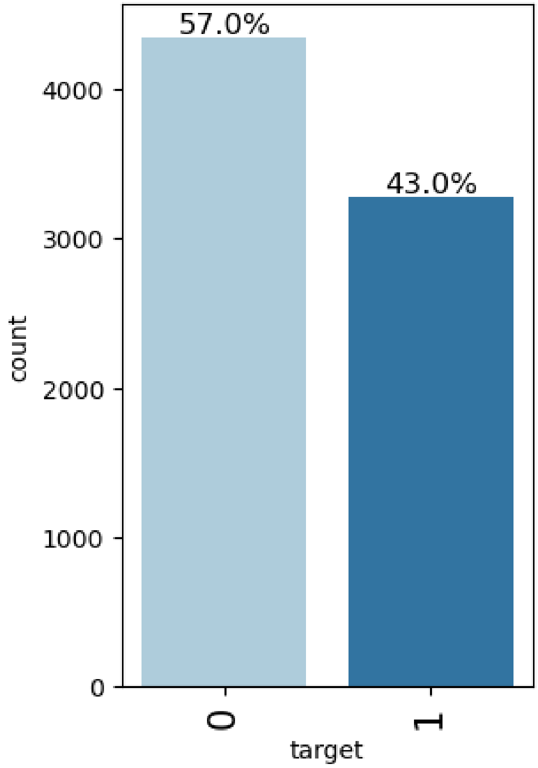
Initial Data Exploration

Function to create labeled barplots

In [202]:

```
def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

In [203]:
```
labeled_barplot(train_clean, 'target', perc=True)
```



The target class in our dataset is characterized by two distinct labels: label 1 signifies tweets that depict genuine disasters, while label 0 encompasses tweets that do not pertain to disasters. Notably, around 56.68% of the total tweets fall into the category of label 0, signifying non-disaster-related content, while the remaining 42.11% belong to label 1, representing tweets describing real disasters.

While there is a slight imbalance in the distribution of these target classes, with class 0 being more prevalent than class 1, it is important to note that this imbalance is not considered a significant concern for our project.

In [204]:

```
# Calculate the top ten values in the 'keyword' column - Train
top_ten_train = train_clean['keyword'].value_counts().head(10).index

# Filter the DataFrame for these values
train_clean_top_ten = train_clean[train_clean['keyword'].isin(top_ten_train)]

# Call the labeled_barplot function
labeled_barplot(data=train_clean_top_ten, feature='keyword', perc=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'keyword'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28080\277826701.py in <module>
      1 # Calculate the top ten values in the 'keyword' column - Train
----> 2 top_ten_train = train_clean['keyword'].value_counts().head(10).index
      3 
      4 # Filter the DataFrame for these values
      5 train_clean_top_ten = train_clean[train_clean['keyword'].isin(top_ten_train)]

~\anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3503             if self.columns.nlevels > 1:
   3504                 return self._getitem_multilevel(key)
-> 3505             indexer = self.columns.get_loc(key)
   3506             if is_integer(indexer):
   3507                 indexer = [indexer]

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
-> 3631                 raise KeyError(key) from err
   3632             except TypeError:
   3633                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 'keyword'
```

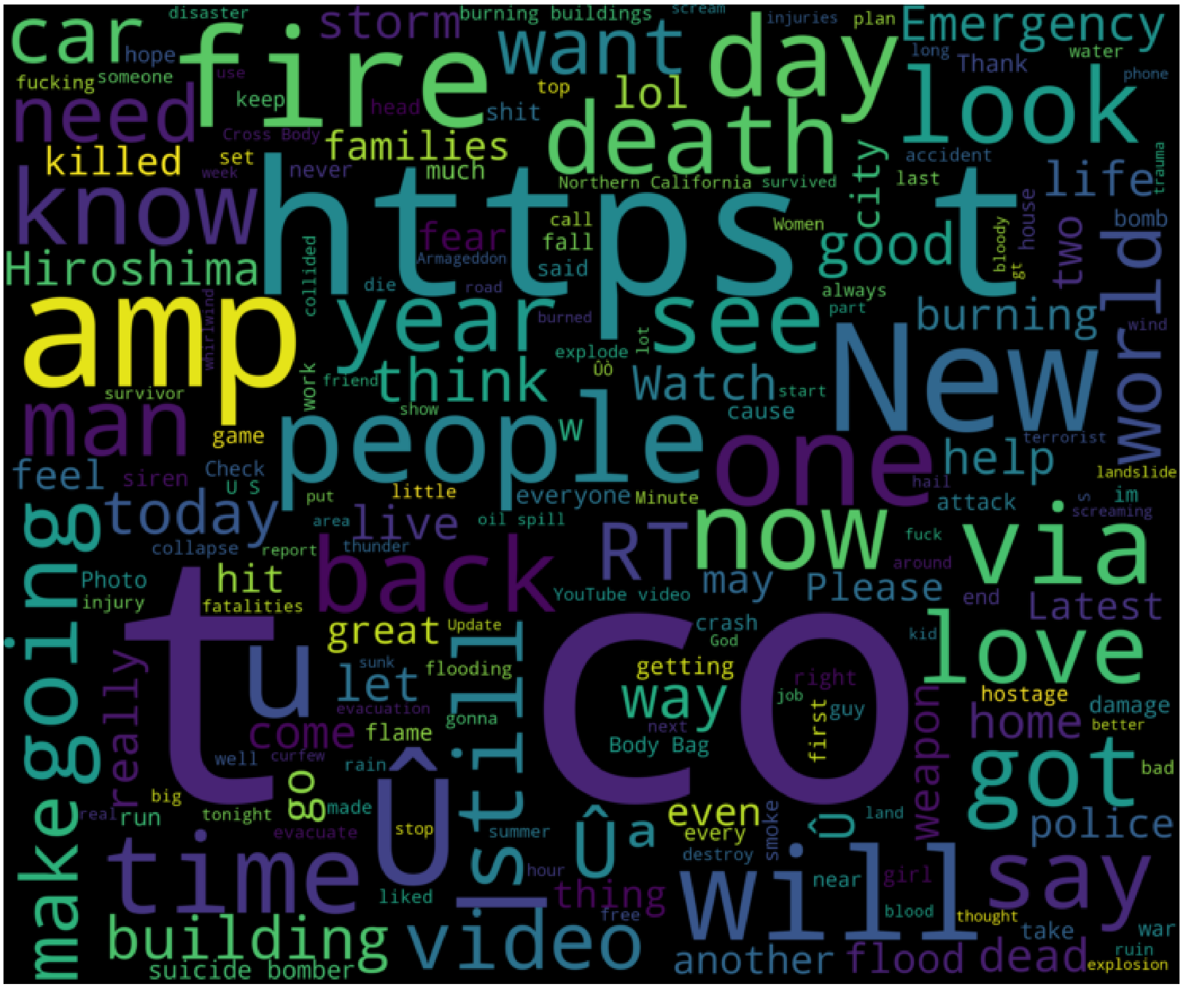
In [205]:
```
# Calculate the top ten values in the 'keyword' column - Test
top_ten_test = test_clean['keyword'].value_counts().head(10).index

# Filter the DataFrame for these values
test_clean_top_ten = test_clean[test_clean['keyword'].isin(top_ten_test)]

# Call the labeled_barplot function
labeled_barplot(data=test_clean_top_ten, feature='keyword', perc=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'keyword'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28080\3881560260.py in <module>
      1 # Calculate the top ten values in the 'keyword' column - Test
----> 2 top_ten_test = test_clean['keyword'].value_counts().head(10).index
      3 
      4 # Filter the DataFrame for these values
      5 test_clean_top_ten = test_clean[test_clean['keyword'].isin(top_ten_test)]

~\anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3503             if self.columns.nlevels > 1:
   3504                 return self._getitem_multilevel(key)
-> 3505             indexer = self.columns.get_loc(key)
   3506             if is_integer(indexer):
   3507                 indexer = [indexer]

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
-> 3631                 raise KeyError(key) from err
   3632             except TypeError:
   3633                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 'keyword'
```

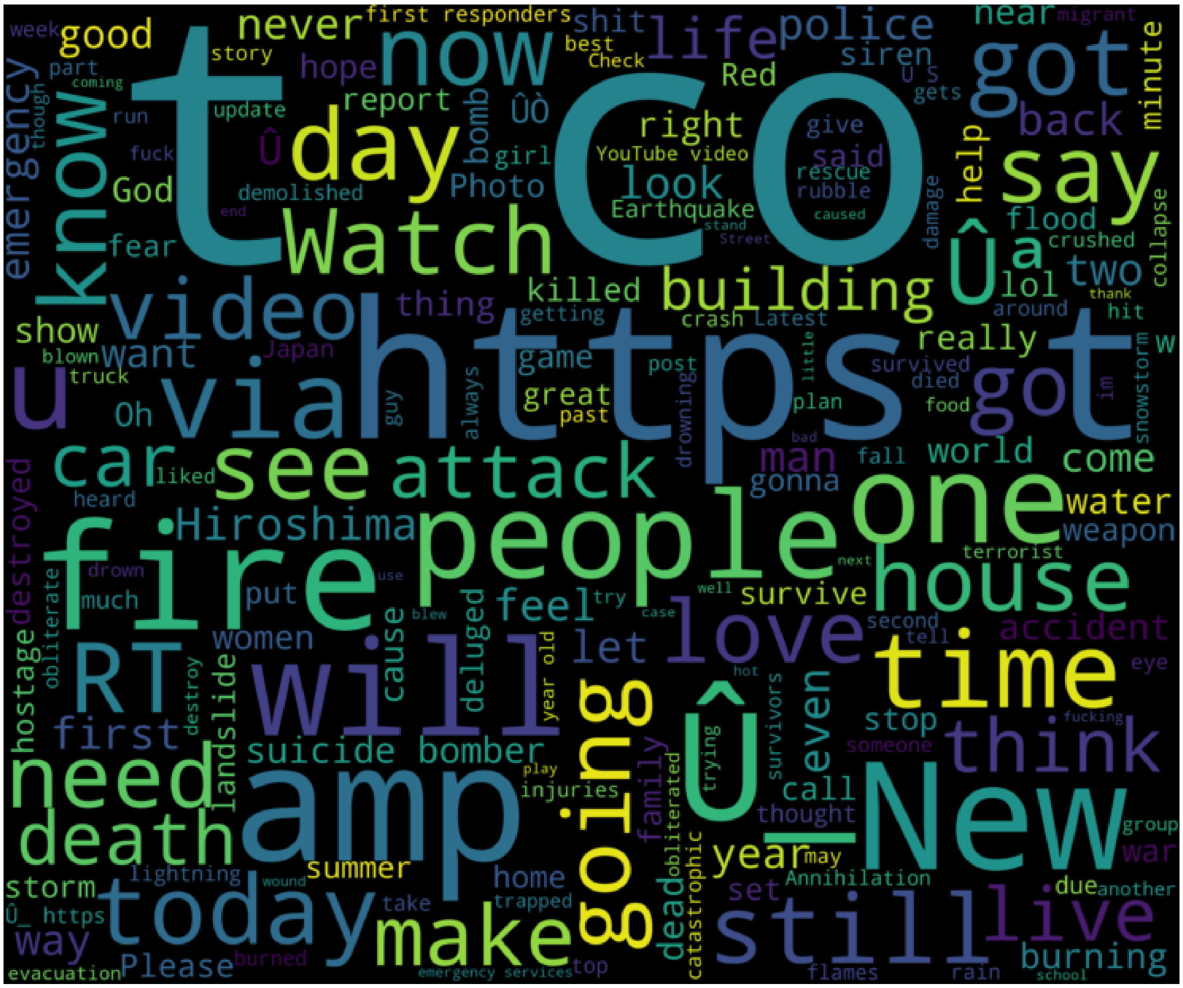
Word Cloud

In [206]:

```
# Generate word cloud for the 'text' column in train_clean DataFrame
wordcloud = WordCloud(stopwords=STOPWORDS,
                      background_color='black',
                      width=3000,
                      height=2500
                     ).generate(' '.join(train_clean['text']))

# Plot the word cloud
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



In [207]:
```
# Generate word cloud for the 'text' column in test_clean DataFrame
wordcloud = WordCloud(stopwords=STOPWORDS,
                      background_color='black',
                      width=3000,
                      height=2500
                     ).generate(' '.join(test_clean['text']))

# Plot the word cloud
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



Data Pre-processing:

- Remove html tags.
- Replace contractions in string. (e.g. replace I'm --> I am) and so on.\
- Remove numbers.
- Tokenization
- Remove Stopwords.
- Lemmatized data
- Use NLTK library to tokenize words , remove stopwords and lemmatize the remaining words.

Remove HTML Tages

In [208]:
```
# Code to remove the html tage
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

train_clean['text'] = train_clean['text'].apply(lambda x: strip_html(x))
train_clean.head()
```

Out[208]:

	id	text	target
0	1	Our Deeds are the Reason of this #earthquake M...	1
1	4	Forest fire near La Ronge Sask. Canada	1
2	5	All residents asked to 'shelter in place' are ...	1
3	6	13,000 people receive #wildfires evacuation or...	1
4	7	Just got sent this photo from Ruby #Alaska as ...	1

Replace contractions in string

In [209]:
```
def replace_contractions(text):
    """Replace contractions in string of text"""
    return contractions.fix(text)

train_clean['text'] = train_clean['text'].apply(lambda x: replace_contractions(x))
train_clean.head()
```

Out[209]:

	id	text	target
0	1	Our Deeds are the Reason of this #earthquake M...	1
1	4	Forest fire near La Ronge Sask. Canada	1
2	5	All residents asked to 'shelter in place' are ...	1
3	6	13,000 people receive #wildfires evacuation or...	1
4	7	Just got sent this photo from Ruby #Alaska as ...	1

Remove numbers

In [210]:
```
def remove_numbers(text):
  text = re.sub(r'\d+', '', text)
  return text

train_clean['text'] = train_clean['text'].apply(lambda x: remove_numbers(x))
train_clean.head()
```

Out[210]:

	id	text	target
0	1	Our Deeds are the Reason of this #earthquake M...	1
1	4	Forest fire near La Ronge Sask. Canada	1
2	5	All residents asked to 'shelter in place' are ...	1
3	6	, people receive #wildfires evacuation orders ...	1
4	7	Just got sent this photo from Ruby #Alaska as ...	1

Apply Tokenization

In [211]:
```
train_clean['text'] = train_clean.apply(lambda row: nltk.word_tokenize(row['text']), axis
train_clean.head()
```

Out[211]:

	id	text	target
0	1	[Our, Deeds, are, the, Reason, of, this, #, ea...	1
1	4	[Forest, fire, near, La, Ronge, Sask, ., Canada]	1
2	5	[All, residents, asked, to, 'shelter, in, plac...	1
3	6	[,, people, receive, #, wildfires, evacuation,...	1
4	7	[Just, got, sent, this, photo, from, Ruby, #, ...	1

Apply Stopwords

In [212]:
```
# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\mulli\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\mulli\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[212]: True

In [213]:
```
from nltk.corpus import stopwords

# Get NLTK English stopwords
stopwords_nltk = set(stopwords.words('english'))

# Define custom stopwords list
custom_stopwords = {"http", "https", '#'}

# Get the count of NLTK stopwords before merging
before_count = len(stopwords_nltk)

# Merge NLTK stopwords with custom stopwords
stopwords_combined = stopwords_nltk.union(custom_stopwords)

# Get the count of combined stopwords after merging
after_count = len(stopwords_combined)

# Print the counts
print("Stopwords count before merging:", before_count)
print("Stopwords count after merging:", after_count)
```

```
Stopwords count before merging: 179
Stopwords count after merging: 182
```

In [214]:
```
train_clean2 = train_clean.copy()
```

In [215]:
```
train_clean2.head()
```

Out[215]:

	id	text	target
0	1	[Our, Deeds, are, the, Reason, of, this, #, ea...	1
1	4	[Forest, fire, near, La, Ronge, Sask, ., Canada]	1
2	5	[All, residents, asked, to, 'shelter, in, plac...	1
3	6	[,, people, receive, #, wildfires, evacuation,...	1
4	7	[Just, got, sent, this, photo, from, Ruby, #, ...	1

In [216]:

```
# Initialize NLTK lemmatizer
lemmatizer = WordNetLemmatizer()


def remove_non_ascii(words):
    """Remove non-ASCII characters from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = unicodedata.normalize('NFKD', word).encode('ascii', 'ignore').decode('u
        new_words.append(new_word)
    return new_words

def to_lowercase(words):
    """Convert all characters to lowercase from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = word.lower()
        new_words.append(new_word)
    return new_words

def remove_punctuation(words):
    """Remove punctuation from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = re.sub(r'[^\w\s]', '', word)
        if new_word != '':
            new_words.append(new_word)
    return new_words

def remove_stopwords(words):
    """Remove stop words from list of tokenized words"""
    new_words = []
    for word in words:
        if word not in stopwords_combined:
            new_words.append(word)
    return new_words

def lemmatize_list(words):
    new_words = []
    for word in words:
        new_words.append(lemmatizer.lemmatize(word, pos='v'))
    return new_words

def normalize(text):
    # Convert to lowercase
    text = text.lower()

    # Remove non-ASCII characters
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8',

    # Remove punctuation
    text = re.sub(r'[^\w\s]', '', text)

    # Tokenize text
    words = text.split()

    # Remove stopwords
    words = [word for word in words if word not in stopwords_combined]

    # Lemmatize words
    words = [lemmatizer.lemmatize(word, pos='v') for word in words]

    # Join words back into a single string
    normalized_text = ' '.join(words)
```

```
    return normalized_text

train_clean2['text'] = train_clean2['text'].apply(lambda x: normalize(' '.join(x)))
train_clean2.head()
```

Out[216]:

	id	text	target
0	1	deeds reason earthquake may allah forgive us	1
1	4	forest fire near la ronge sask canada	1
2	5	residents ask shelter place notify officer eva...	1
3	6	people receive wildfires evacuation order cali...	1
4	7	get send photo ruby alaska smoke wildfires pou...	1

Model 1 - Countervectorizer (Bag of Words)

Build the model based on countvectorizer and Random forest

In [217]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [218]:
```
# Vectorization (Convert text data to numbers).
Count_vec = CountVectorizer(max_features=5000)
data_features = Count_vec.fit_transform(train_clean2['text'])
data_features = data_features.toarray()
```

In [219]:
```
data_features.shape
```

Out[219]: (7613, 5000)

Store Independent and Dependent variables

In [220]:
```
X = data_features
y = train_clean2.target
```

Splitting the Data into Test and Train

In [221]:
```
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.25, random_state=4
```

Random Forest Model

In [222]:
```
# Using Random Forest to build model for the classification of reviews.
forest = RandomForestClassifier(n_estimators=10, n_jobs=4)
forest = forest.fit(X_train, y_train)
print(forest)
print(np.mean(cross_val_score(forest, X, y, cv=10)))
```

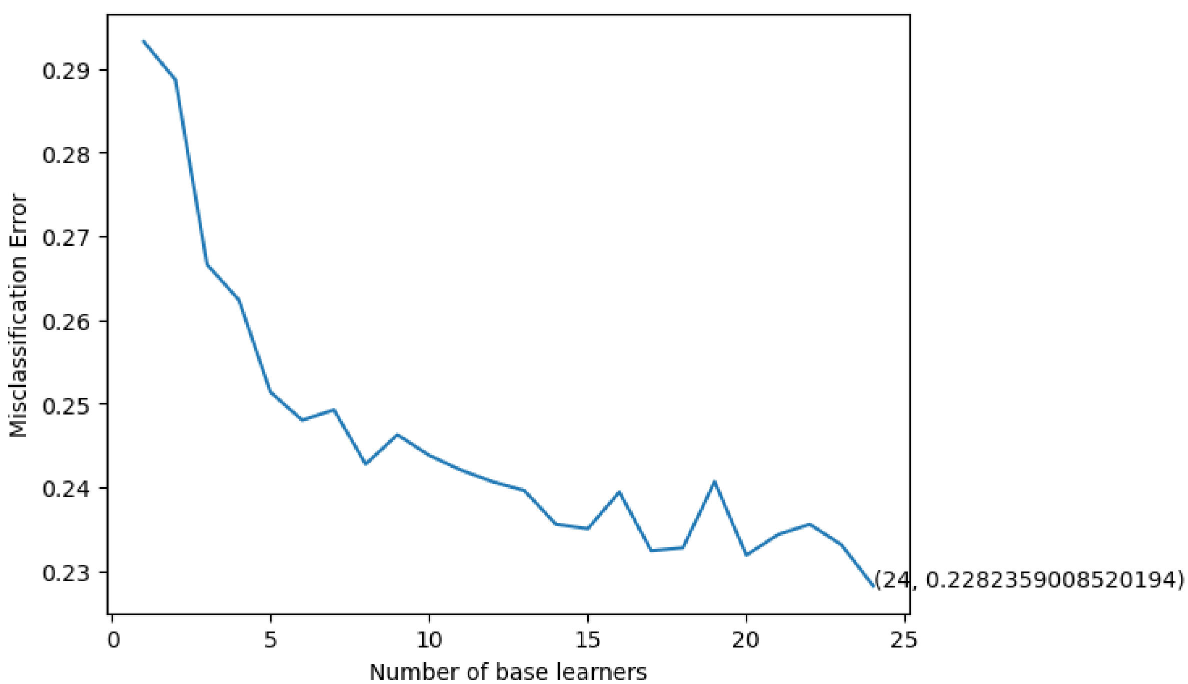
```
RandomForestClassifier(n_estimators=10, n_jobs=4)
0.5797958895085552
```

Optimize the parameter: The number of trees in the random forest model(n_estimators)

In [223]:
```
# Finding optimal number of base learners using k-fold CV ->
base_ln = [x for x in range(1, 25)]
```

In [224]:
```
# K-Fold Cross - validation .
cv_scores = []
for b in base_ln:
    clf = RandomForestClassifier(n_estimators = b)
    scores = cross_val_score(clf, X_train, y_train, cv = 5, scoring = 'accuracy')
    cv_scores.append(scores.mean())
```

In [225]:
```
# plot the error as k increases
error = [1 - x for x in cv_scores]
optimal_learners = base_ln[error.index(min(error))]
plt.plot(base_ln, error)
xy = (optimal_learners, min(error))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.xlabel("Number of base learners")
plt.ylabel("Misclassification Error")
plt.show()
```



Observations:

- Misclassification error seems to be flattening out after 10 k-folds
- At 22 base learners, the misclassification error is about 22.8%

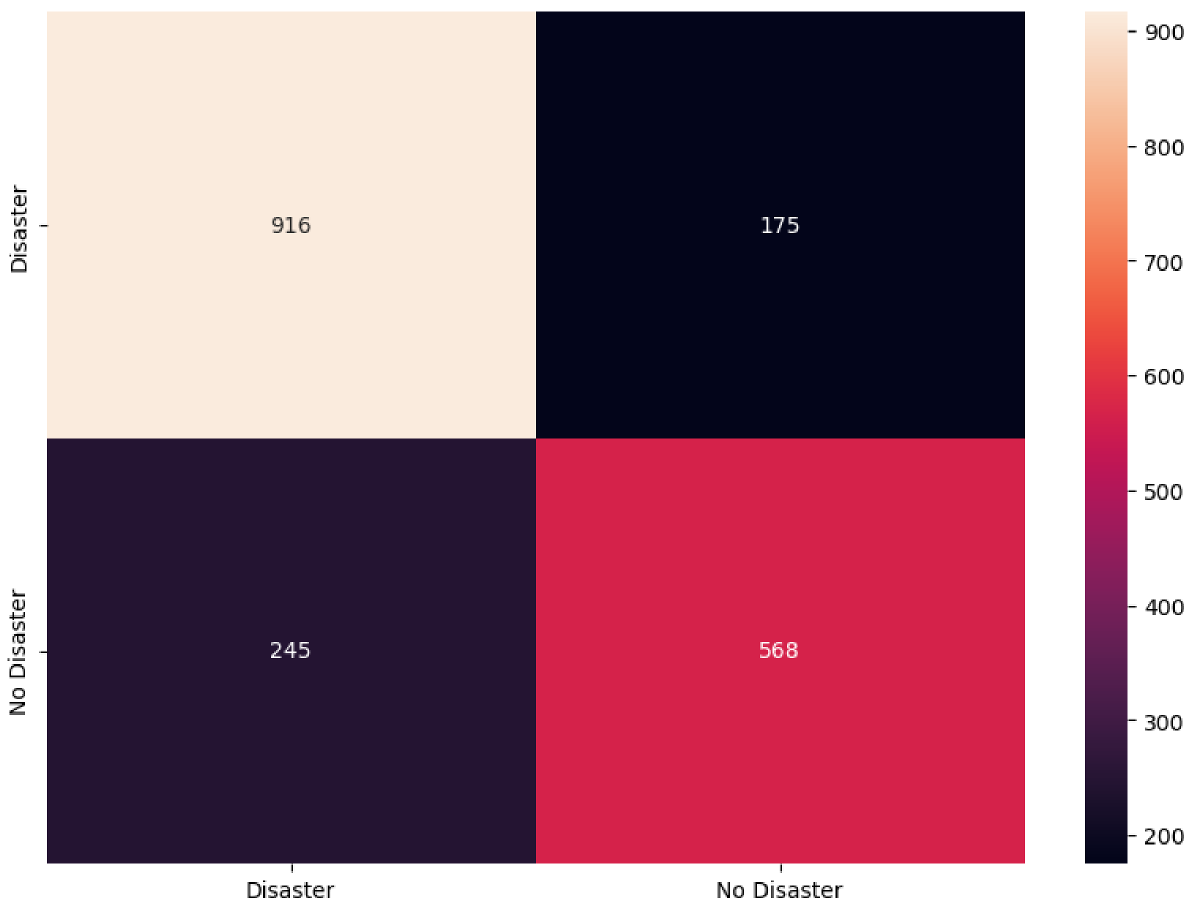
In [226]:
```
# Train the best model and calculating accuracy on test data .
clf = RandomForestClassifier(n_estimators = optimal_learners) # Initialize the Random
clf.fit(X_train, y_train) # Fit the classifer on
clf.score(X_test, y_test)
```

Out[226]: 0.7794117647058824

In [227]:
```
# Predict the result for test data using the model built above.
result = clf.predict(X_test)
```

In [228]:
```
# Plot the confusion matrix

conf_mat = confusion_matrix(y_test, result)

df_cm = pd.DataFrame(conf_mat, index = [i for i in ['Disaster', 'No Disaster']],
                columns = [i for i in ['Disaster', 'No Disaster']])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

Out[228]: <AxesSubplot:>



Observations:

- 903 Disaster sentiments were classified correctly
- 554 No Disaster were classified correctly

In [229]:
```
# Plotting the classification report
cr=metrics.classification_report(y_test, result)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.79      0.84      0.81      1091
           1       0.76      0.70      0.73       813

    accuracy                           0.78      1904
   macro avg       0.78      0.77      0.77      1904
weighted avg       0.78      0.78      0.78      1904
```

Observations:

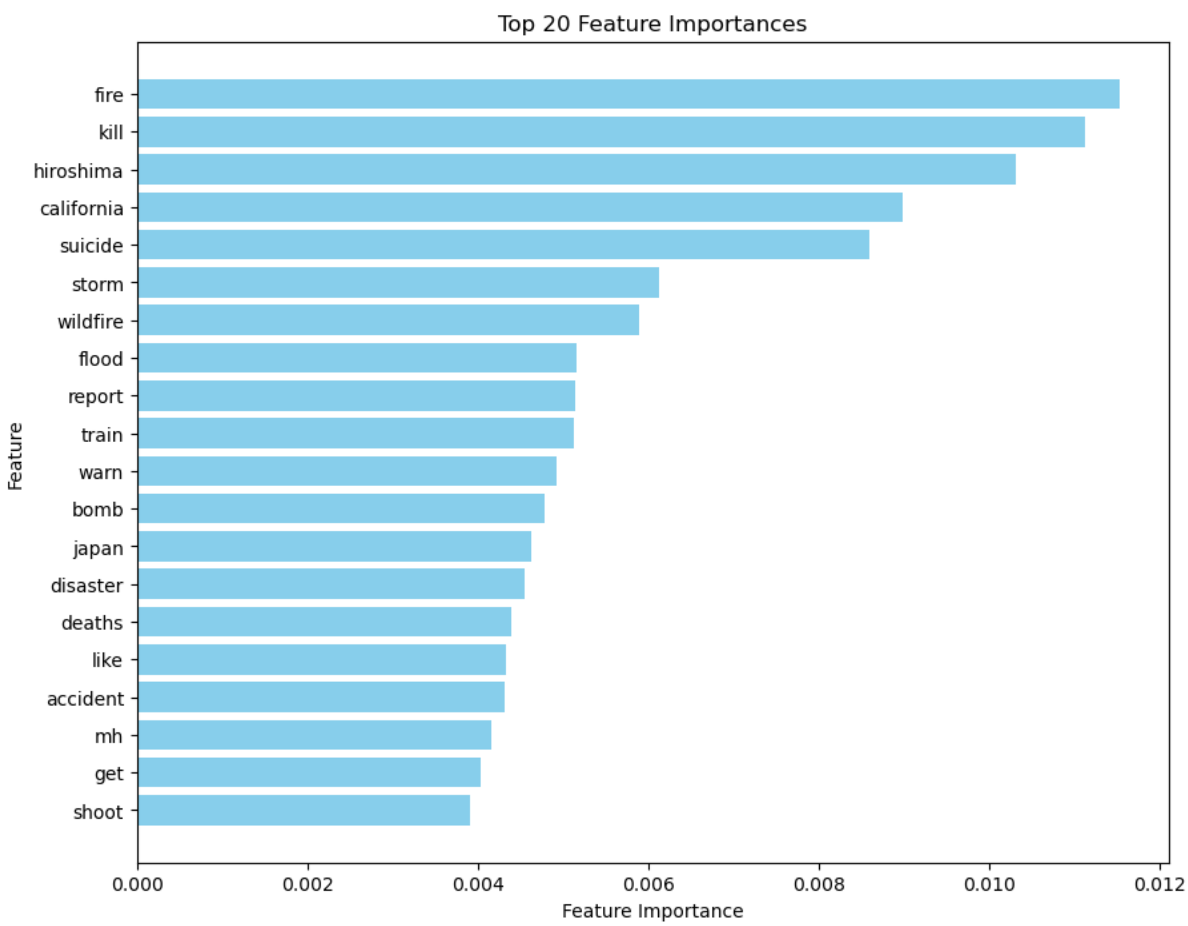
- Overall accuracy is about 77%
- Disasters sentiments seem to be misclassified the most (False Negative)
- Accuracy and F-1 scores can be improved upon with better models

NOTE: Although not stated in the problem, it will be assumed that False Negatives are more costly than False Positives. In other words, if no disaster is predicted and no preparation or prevention measures are put into place and a disaster DOES occur, this is the worst case scenario. It is more costly than a False Positive, where a disaster is predicted, measures are taken, but no disaster materializes. Therefore, we will focus on minimizing False Negatives.

Feature Importance and Wordcloud of top 20 words from countvectorizer+Randomforest based model

In [230]:

```
# Extract top 20 features and their importances
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
top_features = [all_features[i] for i in indices[:20]]  # Select top 20 features
top_importances = importances[indices[:20]]  # Corresponding importances

# Plot the feature importances
plt.figure(figsize=(10, 8))
plt.barh(range(len(top_features)), top_importances, color='skyblue')
plt.yticks(range(len(top_features)), top_features)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Top 20 Feature Importances')
plt.gca().invert_yaxis()  # Invert y-axis to display highest importance at the top
plt.show()
```



In [231]:

```
# Concatenate the top 20 features into a single string
wordcloud_text = ' '.join(top_features)

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(wordcloud_

# Plot the word cloud
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Top 20 Features Word Cloud')
plt.axis("off")
plt.show()
```

Top 20 Features Word Cloud

Conclusions for Model 1

- Overall accuracy of the model: 77%
- Did a better job classifying Disasters than No Disasters

Final word cloud contains key words that can help law enforcement and other first responders identify potential disasters sooner.

Model 2: Term Frequency(TF) - Inverse Document Frequency(IDF)

In [232]:

```
# Using TfidfVectorizer to convert text data to numbers.

tfidf_vect = TfidfVectorizer(max_features=5000)
data_features = tfidf_vect.fit_transform(train_clean2['text'])

data_features = data_features.toarray()
```

In [233]:
```
data_features.shape
```
Out[233]:
```
(7613, 5000)
```

Store Independent and Dependent variables

In [234]:
```
X = data_features
y = train_clean2.target
```

Split the data into train and test

In [235]:
```
X_train, X_test, y_train, y_test =train_test_split (X, y, test_size=0.25, random_state=42
```

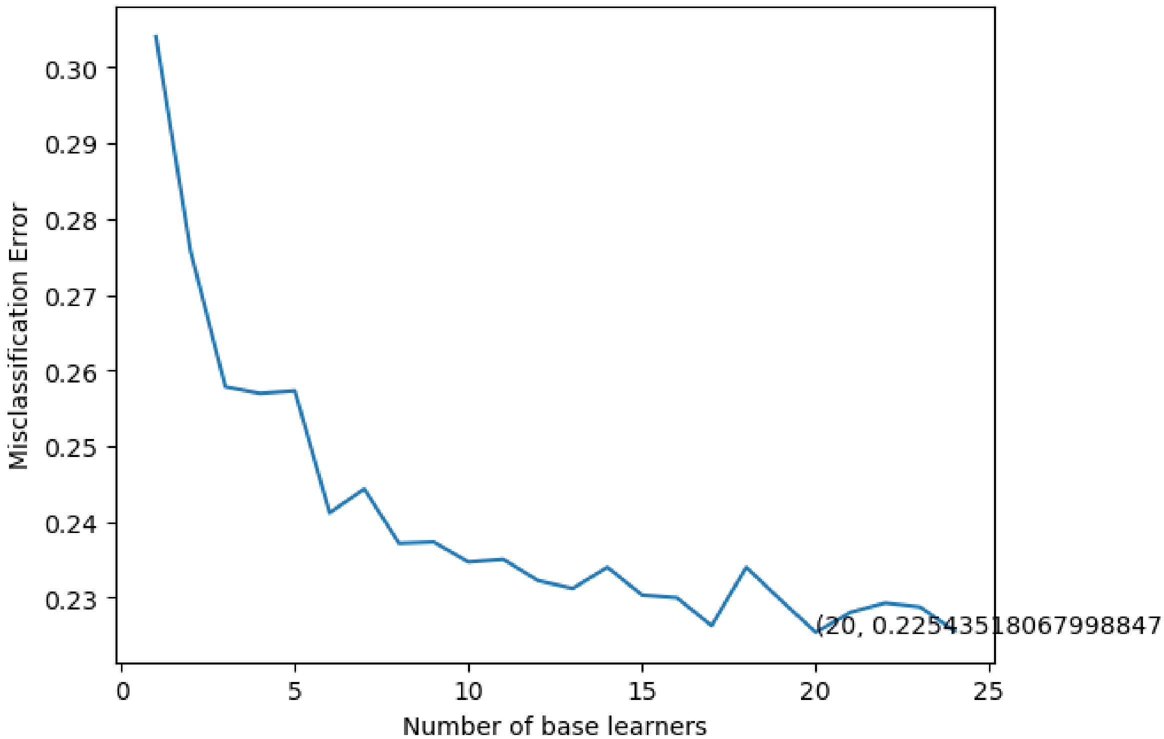
Random Forest Model

In [236]:
```
# Using Random Forest to build model for the classification of reviews.

forest = RandomForestClassifier(n_estimators=10, n_jobs=4)
forest = forest.fit(X_train, y_train)
print(forest)
print(np.mean(cross_val_score(forest, X, y, cv=10)))
```
```
RandomForestClassifier(n_estimators=10, n_jobs=4)
0.5706031571940499
```

In [237]:
```
# Finding optimal number of base learners using k-fold CV ->
base_ln = [x for x in range(1, 25)]
```

In [238]:
```
# K-Fold Cross - validation .
cv_scores = []
for b in base_ln:
    clf = RandomForestClassifier(n_estimators = b)
    scores = cross_val_score(clf, X_train, y_train, cv = 5, scoring = 'accuracy')
    cv_scores.append(scores.mean())
```

In [239]:
```
# Plot the misclassification error for each of estimators
error = [1 - x for x in cv_scores]
optimal_learners = base_ln[error.index(min(error))]
plt.plot(base_ln, error)
xy = (optimal_learners, min(error))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.xlabel("Number of base learners")
plt.ylabel("Misclassification Error")
plt.show()
```



Observations:

- Misclassification error seems to be flattening out after 20 k-folds but there is still minor fluctuations. It might be prudent to extend the number of base learners to 30 or even 50 to be sure.
- At 23 base learners, the misclassification error is about 22.6%

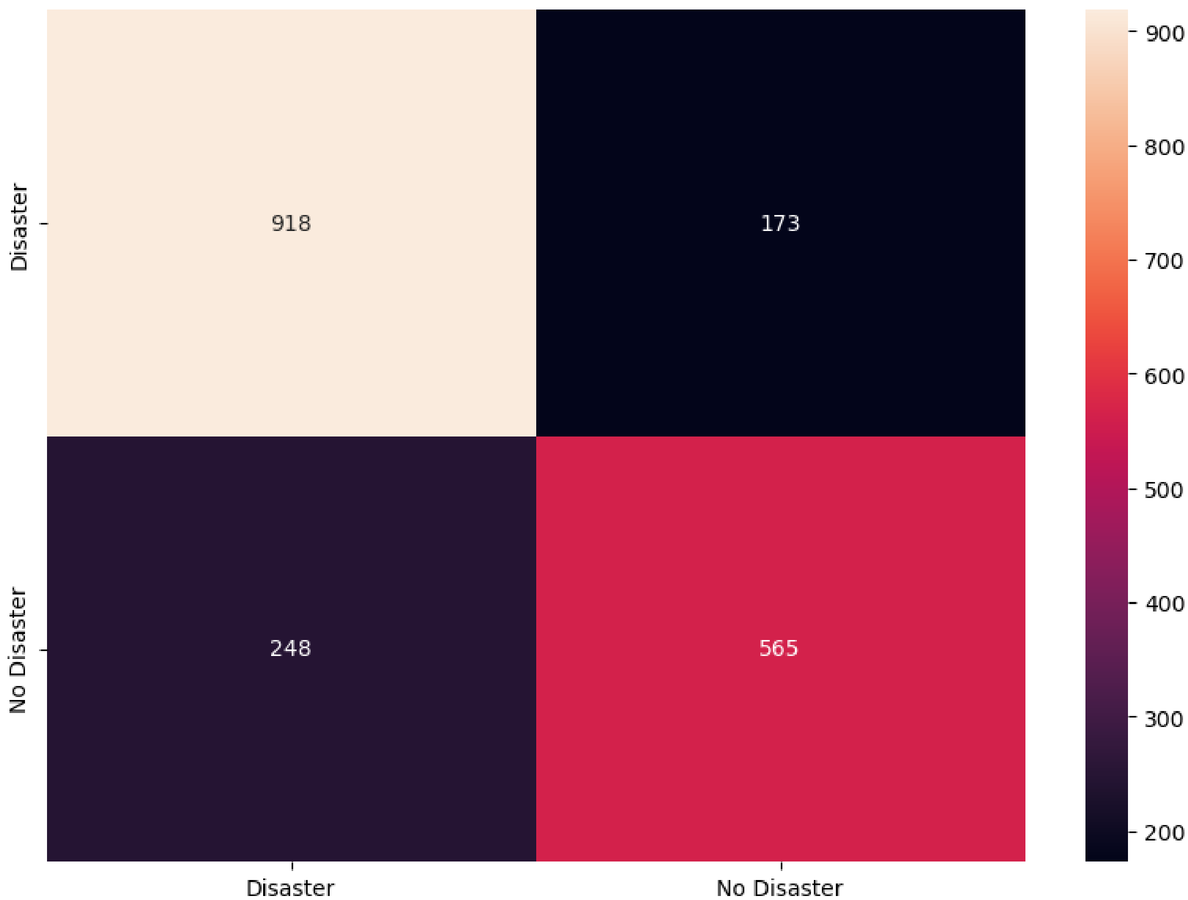
In [240]:
```
# Train the best model and calculating accuracy on test data .
clf = RandomForestClassifier(n_estimators = optimal_learners)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

Out[240]: 0.7788865546218487

In [241]:
```
# Predict the result for test data using the model built above.
result = clf.predict(X_test)
```

In [242]:
```
# Plot the confusion matrix
conf_mat = confusion_matrix(y_test, result)

df_cm = pd.DataFrame(conf_mat, index = [i for i in ['Disaster', 'No Disaster']],
                columns = [i for i in ['Disaster', 'No Disaster']])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

Out[242]: <AxesSubplot:>



Observations:

- 926 Disaster sentiments were classified correctly
- 557 No Disaster were classified correctly

In [243]:
```
# Plotting the classification report
cr=metrics.classification_report(y_test, result)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.79      0.84      0.81      1091
           1       0.77      0.69      0.73       813

    accuracy                           0.78      1904
   macro avg       0.78      0.77      0.77      1904
weighted avg       0.78      0.78      0.78      1904
```

Observations:

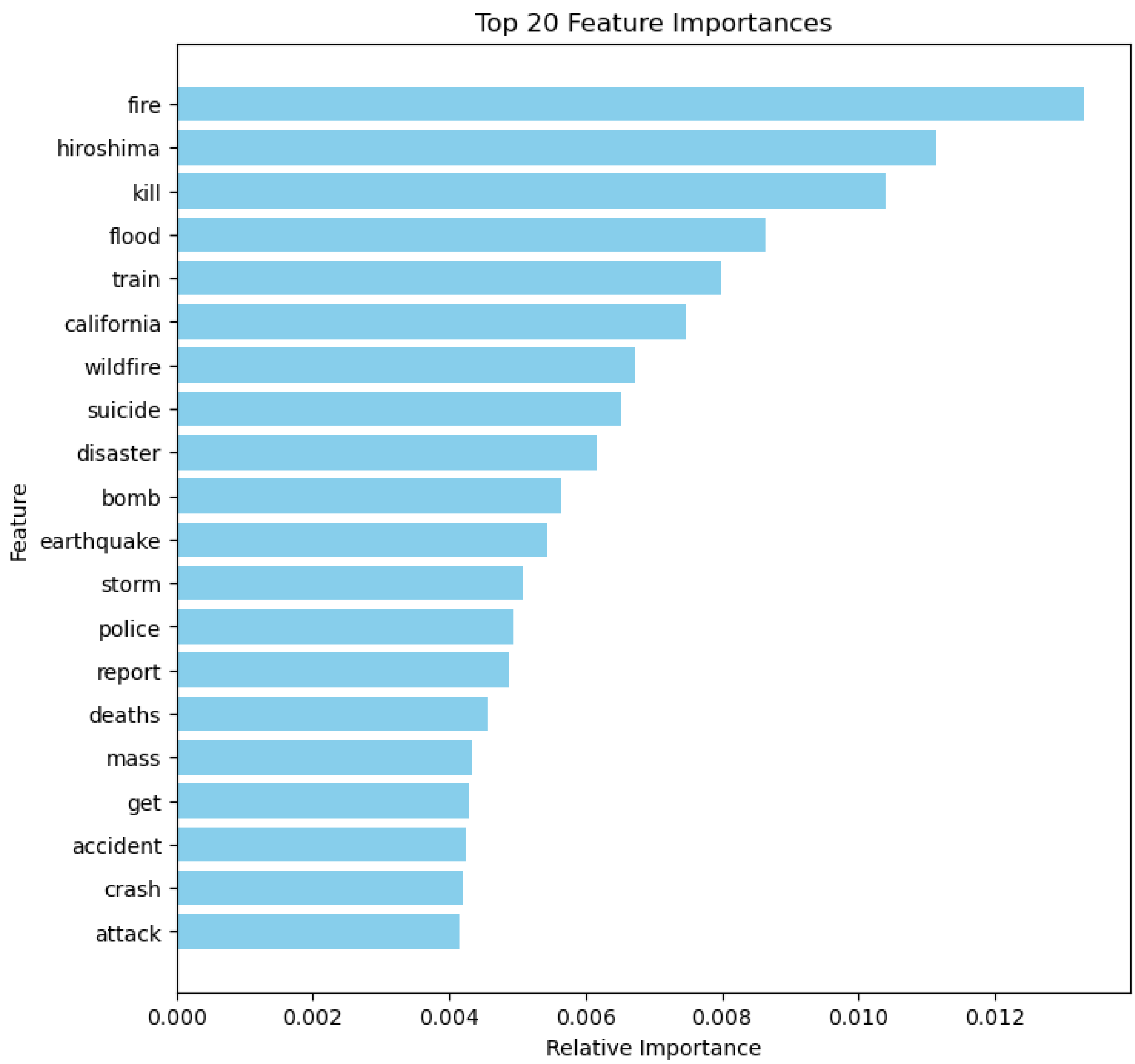
- Overall accuracy is about 78%
- Recall score went up for No Disaster (from 83% to 85%)
- However, recall score went down for Disaster (from 68% to 69%)

The accuracy of the model went up, but I would prefer a model that maximizes the Recall scores. We may have to try an additional model.

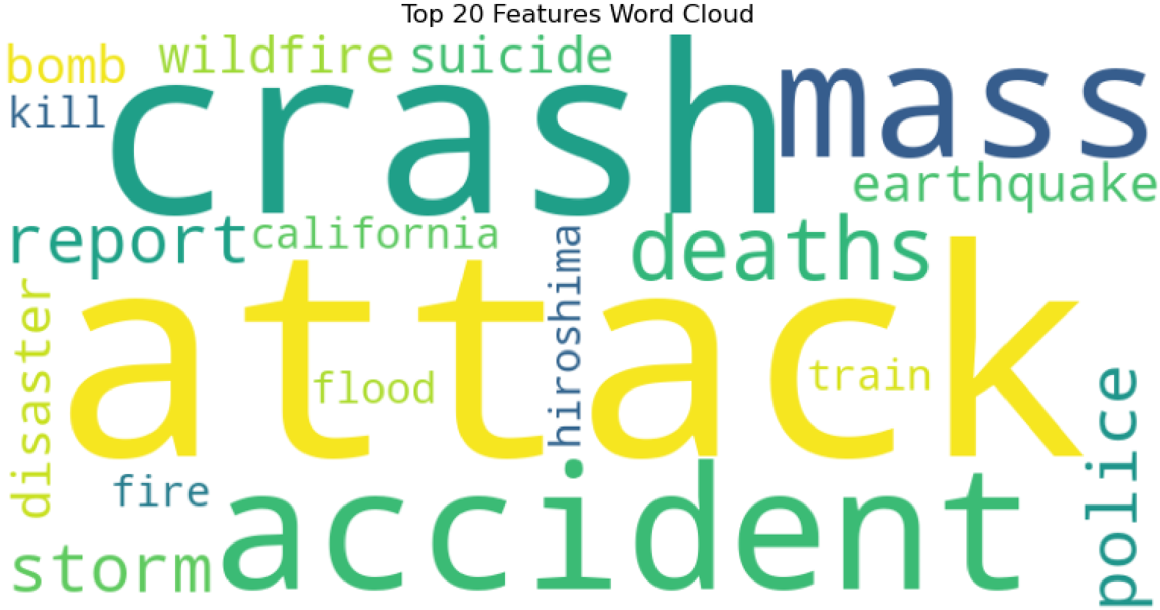
Feature Importance and Wordcloud of top 20 words from TF-IDF model

In [244]:

```
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]  # Sort indices in descending order

# Select only the top 20 features
top_indices = indices[:20]
top_importances = importances[top_indices][::-1]  # Reverse to plot in descending order
top_features = [all_features[i] for i in top_indices][::-1]  # Reverse to match importance

plt.figure(figsize=(8, 8))
plt.title('Top 20 Feature Importances')
plt.barh(range(len(top_indices)), top_importances, color='skyblue', align='center')
plt.yticks(range(len(top_indices)), top_features)
plt.xlabel('Relative Importance')
plt.ylabel('Feature')
plt.show()
```



In [245]:
```
from wordcloud import WordCloud

# Concatenate the top 20 features into a single string
wordcloud_text = ' '.join(top_features)

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(wordcloud_

# Plot the word cloud
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Top 20 Features Word Cloud')
plt.axis("off")
plt.show()
```



Conclusions for Model 2

- Overall accuracy of the model: 78%
- Did a better job classifying Disasters than No Disasters

Final word cloud contains key words that can help law enforcement and other first responders identify potential disasters sooner.

Model 3: Word2Vec

- **Word2vec is a group of shallow, two-layer neural networks** that are trained to represent the linguistic and contextual similarity of words through numbers based on their semantics as received from the corpus of text it is trained on.
- Word2vec takes as its input a large corpus or document of text and produces a vector space. The choice for the dimensionality of the vector space /embedding vector is typically a few hundreds. Each unique word in the document is assigned a corresponding vector in the space.

In [76]:
```
words_list =[]
for i in train_clean2['text']:

 li = list(i.split(" "))

 words_list.append(li)
```

In [77]:
```
# Let's have a look into words_list
words_list[0:5]
```

Out[77]:
```
[['deeds', 'reason', 'earthquake', 'may', 'allah', 'forgive', 'us'],
 ['forest', 'fire', 'near', 'la', 'ronge', 'sask', 'canada'],
 ['residents',
  'ask',
  'shelter',
  'place',
  'notify',
  'officer',
  'evacuation',
  'shelter',
  'place',
  'order',
  'expect'],
 ['people', 'receive', 'wildfires', 'evacuation', 'order', 'california'],
 ['get',
  'send',
  'photo',
  'ruby',
  'alaska',
  'smoke',
  'wildfires',
  'pour',
  'school']]
```

In [78]:
```
import gensim
from gensim.models import Word2Vec
from tqdm import tqdm
print(gensim.__version__)
```

```
4.1.2
```

In [79]:
```
# Model creation
model= Word2Vec(words_list, min_count = 1, workers = 4)
```

In [80]:
```
# saving the model
model.save("word2vec.model")
```

In [81]:
```
words = model.wv.key_to_index
vocab_size = len(words)
len(words)
```

Out[81]:
```
19464
```

In [82]:
```
word = "fire"
model.wv[word]
```

Out[82]:
```
array([-0.0259034 ,  0.20709544,  0.05028643, -0.02640318,  0.05395012,
       -0.27942955,  0.04324514,  0.42633265, -0.06721286, -0.04342008,
       -0.12659079, -0.24616338,  0.01909613,  0.14441855,  0.03263763,
       -0.14685422,  0.04305409, -0.15830684,  0.04694482, -0.4205667 ,
        0.04242992,  0.14663969,  0.10048539, -0.01881831, -0.03029871,
        0.08001149, -0.12407713,  0.01994655, -0.18696626,  0.07625499,
        0.15630087, -0.0312388 , -0.05865996, -0.18239667, -0.10987622,
        0.17593132, -0.00663154, -0.10547341, -0.13350286, -0.31266072,
        0.00740254, -0.15784411, -0.14970224,  0.03971724,  0.10806606,
       -0.05782164, -0.09506229, -0.11967311,  0.09577966,  0.10231493,
        0.06945758, -0.20114346, -0.13659546, -0.04243094, -0.12446079,
        0.02648102,  0.02551899,  0.0089324 , -0.1395797 ,  0.02734625,
        0.05633428,  0.03908587,  0.07215782,  0.09310938, -0.22034992,
        0.20575619,  0.1247687 ,  0.10245593, -0.22852027,  0.2593635 ,
       -0.1051332 ,  0.07878345,  0.10855293, -0.13389939,  0.10565499,
        0.10857384,  0.08707609, -0.06077725, -0.12543792,  0.0750083 ,
       -0.11006029, -0.01381307, -0.21797748,  0.26679373, -0.09651249,
       -0.04016635,  0.04498596,  0.13630079,  0.12587275,  0.09292441,
        0.21705244,  0.04805683,  0.1366745 , -0.00476816,  0.35600054,
        0.16293012,  0.2186903 , -0.19340974,  0.09294153,  0.04687116],
      dtype=float32)
```

In [83]:
```
# Top 10 similar words to the word 'kill'
similar = model.wv.similar_by_word('kill')
print(similar)
```

```
[('people', 0.9959092140197754), ('via', 0.9951726198196411), ('dead', 0.995143294334411
6), ('come', 0.9948434233665466), ('go', 0.9947243332862854), ('get', 0.99436938762664
8), ('say', 0.9943189024925232), ('like', 0.9942993521690369), ('would', 0.9942784309387
207), ('crash', 0.9942275881767273)]
```

In [84]:
```
def average_word_vectors(words, model, vocabulary, num_features):
    feature_vector = np.zeros((num_features,), dtype="float64")
    nwords = 0.

    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = np.add(feature_vector, model.wv.get_vector(word))

    if nwords:
        feature_vector = np.divide(feature_vector, nwords)

    return feature_vector

def averaged_word_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.key_to_index.keys())

    features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_features)
                for tokenized_sentence in corpus]

    return np.array(features)

feature_size = 100
# get document level embeddings
w2v_feature_array = averaged_word_vectorizer(corpus=words_list, model=model,
                                             num_features=feature_size)
pd.DataFrame(w2v_feature_array)
```

Out[84]:

	0	1	2	3	4	5	6	7	8	9
0	-0.004144	0.042119	0.013041	-0.001919	0.007262	-0.068290	0.013447	0.097227	-0.008790	-0.013756
1	-0.006636	0.052613	0.015474	-0.006226	0.012324	-0.067307	0.009941	0.104635	-0.017722	-0.013209
2	-0.002144	0.023500	0.008214	-0.001104	0.007316	-0.035120	0.006311	0.051432	-0.006055	-0.008717
3	-0.008206	0.064465	0.018468	-0.010327	0.012547	-0.092402	0.016217	0.127211	-0.018415	-0.017397
4	-0.010279	0.046908	0.012578	-0.004827	0.008255	-0.068652	0.015394	0.100302	-0.016210	-0.013588
...	...	...	...	...	...	...	...	...	...	...
7608	-0.006603	0.052678	0.011931	-0.006747	0.010889	-0.078311	0.015038	0.109035	-0.013120	-0.013023
7609	-0.005710	0.060513	0.018224	-0.007609	0.013454	-0.080764	0.010029	0.118021	-0.021607	-0.013756
7610	-0.004886	0.015189	-0.000425	-0.002707	0.008225	-0.026810	0.004891	0.043975	-0.005284	-0.007423
7611	-0.004826	0.031616	0.008863	-0.002964	0.007544	-0.044652	0.008724	0.066054	-0.009526	-0.007267
7612	-0.010793	0.073862	0.023097	-0.011634	0.014584	-0.095210	0.013594	0.136752	-0.021455	-0.015591

7613 rows × 100 columns

In [85]:
```
X = w2v_feature_array
y = train_clean2.target
```

In [86]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25, random_state=42)
```

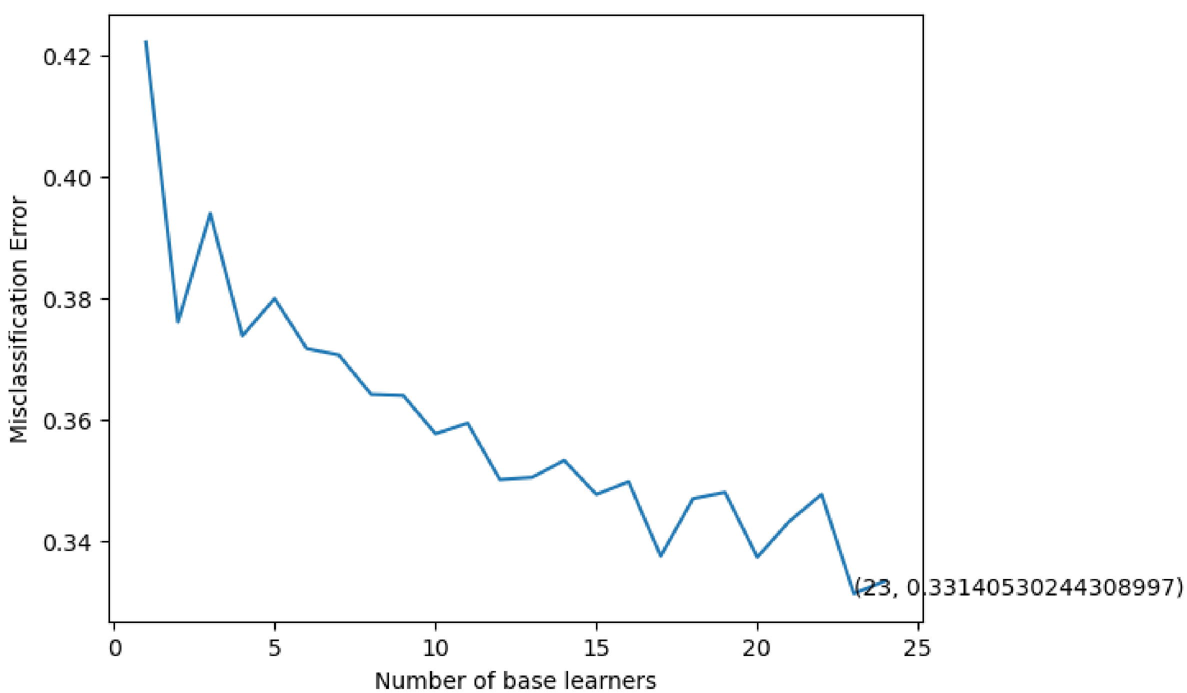
In [87]:
```
# Using Random Forest to build model for the classification of reviews.

forest = RandomForestClassifier(n_estimators=10, n_jobs=4)
forest = forest.fit(X_train, y_train)
print(forest)
print(np.mean(cross_val_score(forest, X, y, cv=10)))
```

```
RandomForestClassifier(n_estimators=10, n_jobs=4)
0.5917534946765033
```

In [88]:
```
# Finding optimal number of base learners using k-fold CV ->
base_ln = [x for x in range(1, 25)]
```

In [89]:
```
#  K-Fold Cross - validation .
cv_scores = [] # Initializi
for b in base_ln:
    clf = RandomForestClassifier(n_estimators = b)
    scores = cross_val_score(clf, X_train, y_train, cv = 5, scoring = 'accuracy')
    cv_scores.append(scores.mean())
```

In [90]:
```
# Plot the misclassification error for each of estimators
error = [1 - x for x in cv_scores]
optimal_learners = base_ln[error.index(min(error))]
plt.plot(base_ln, error)
xy = (optimal_learners, min(error))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.xlabel("Number of base learners")
plt.ylabel("Misclassification Error")
plt.show()
```



Observations:

- Misclassification error does not seems to be flattening out after 20+ k-folds. There are still minor fluctuations.
- At 23 base learners, the misclassification error is about 33.1%

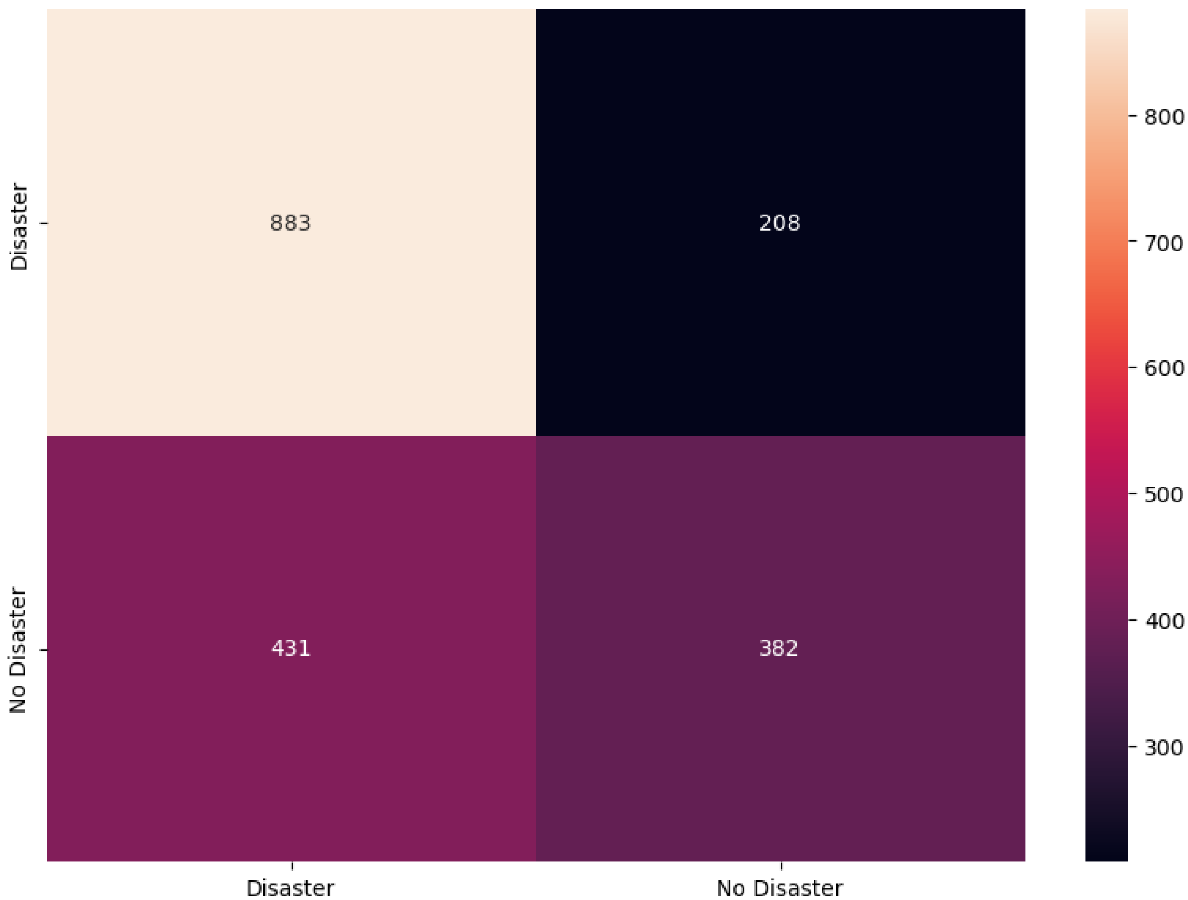
In [91]:
```
# Train the best model and calculating accuracy on test data .
clf = RandomForestClassifier(n_estimators = optimal_learners)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

Out[91]: 0.664390756302521

In [92]:
```
# Predict the result for test data using the model built above.
result = clf.predict(X_test)
```

In [93]:
```
# Plot the confusion matrix
conf_mat = confusion_matrix(y_test, result)

df_cm = pd.DataFrame(conf_mat, index = [i for i in ['Disaster', 'No Disaster']],
                columns = [i for i in ['Disaster', 'No Disaster']])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

Out[93]: <AxesSubplot:>



Observations:

- 887 Disaster tweets were classified correctly
- 382 No Disaster tweets were classified correctly

This model had lower classification efficiency than either of the previous 2 models: Disaster tweets and No Disaster tweets are bot3 significantly lower

In [94]:
```
#  Plotting the classification report
cr=metrics.classification_report(y_test, result) # From Project 2
print(cr)
```

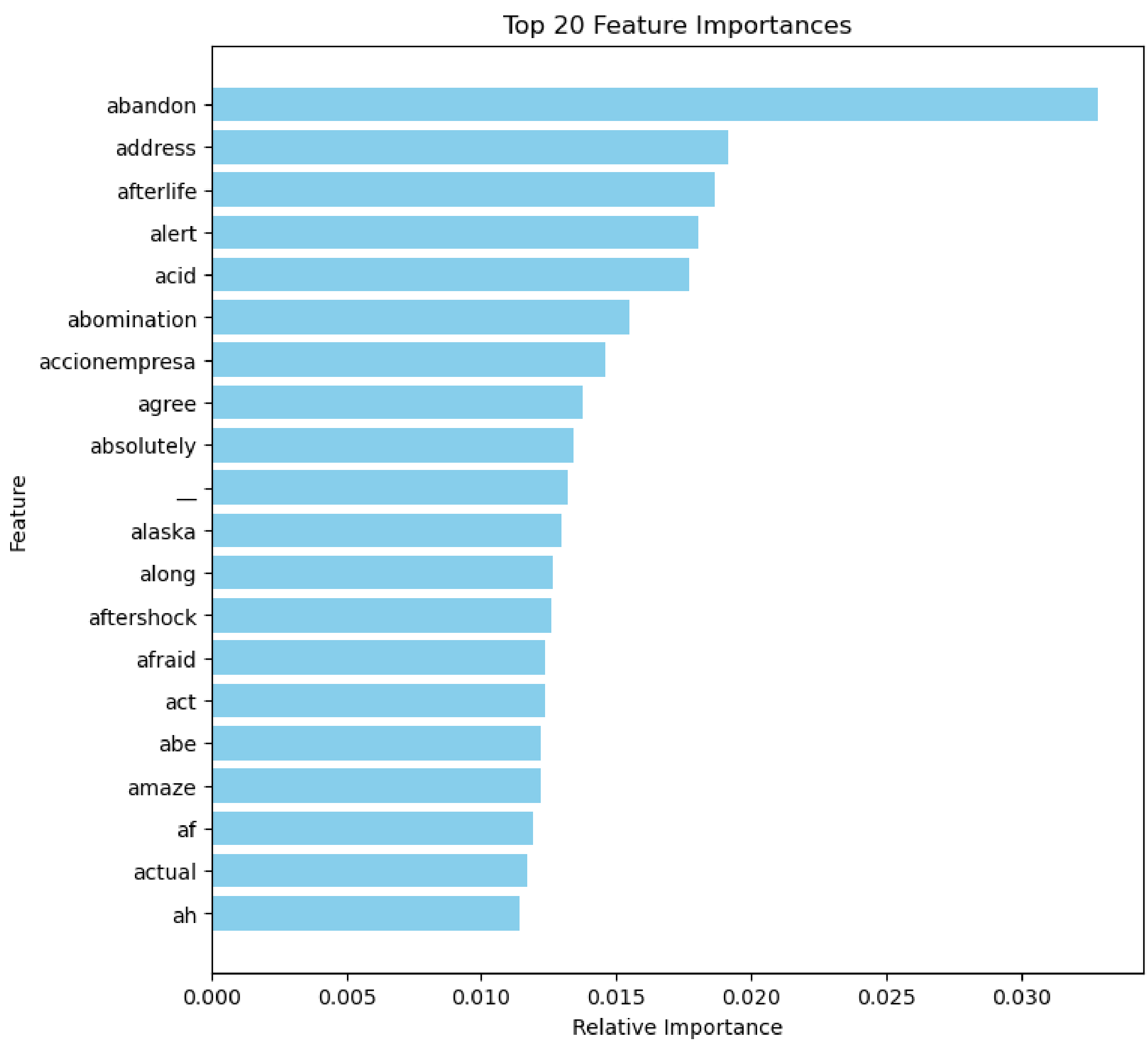
```
              precision    recall  f1-score   support

           0       0.67      0.81      0.73      1091
           1       0.65      0.47      0.54       813

    accuracy                           0.66      1904
   macro avg       0.66      0.64      0.64      1904
weighted avg       0.66      0.66      0.65      1904
```

Observations:

- Overall accuracy is about 66%
- False Negatives almost double
- Accuracy and Recall scores are lower than our first two models

Feature Importance and Wordcloud of top 20 important features from Word2Vec+Randomforest based mode

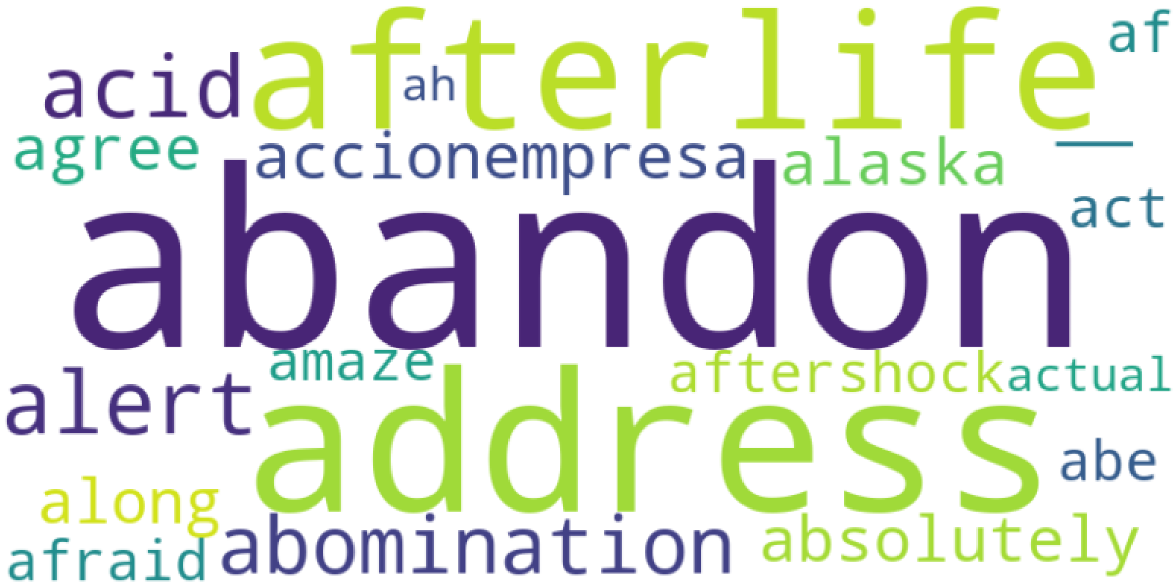
In [106]:
```
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]  # Sort indices in descending order

# Select only the top 20 features
top_indices = indices[:20]
top_importances = importances[top_indices][::-1]  # Reverse to plot in descending order
top_features = [all_features[i] for i in top_indices][::-1]  # Reverse to match importanc

plt.figure(figsize=(8, 8))
plt.title('Top 20 Feature Importances')
plt.barh(range(len(top_indices)), top_importances, color='skyblue', align='center')
plt.yticks(range(len(top_indices)), top_features)
plt.xlabel('Relative Importance')
plt.ylabel('Feature')
plt.show()
```



In [107]:

```
# Create a dictionary with words as keys and importances as values
word_importance_dict = {word: importance for word, importance in zip(top_features, top_im

# Generate the word cloud with importance-based sizing
wordcloud = WordCloud(width=800, height=400, background_color='white', prefer_horizontal=

# Plot the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud of Top 20 Features')
plt.axis("off")
plt.show()
```

Word Cloud of Top 20 Features



Conclusions for Model 3

- Overall accuracy of the model: 66%
- Recall scores dropped to 81% and 47%, respectively.
- THe important features showed all words starting with the letter A
- The model does an inferior job classifying Disasters and False Negatives

This model did a much worse job than either of the previous two models

Model 4: Modeling using TFIDF weighted Word2Vec

In [130]:
```
i = 0
list_of_sentance = []
for sentence in train_clean2['text']:
    list_of_sentance.append(sentence.split())

w2v_model = Word2Vec(list_of_sentance, min_count=1, vector_size=300, workers=4)
w2v_words = list(w2v_model.wv.index_to_key)
```

In [136]:
```
# Initialize and fit the TfidfVectorizer
tfidf_model = TfidfVectorizer()
tfidf_model.fit(train_clean2['text'].tolist())

# Get the vocabulary and IDF values
dictionary = dict(zip(tfidf_model.get_feature_names_out(), tfidf_model.idf_))
tfidf_words = set(tfidf_model.get_feature_names_out())
```

In [137]:
```
tfidf_w2v_vectors = []  # List to store the TF-IDF weighted Word2Vec vectors
for sentence in tqdm(train_clean2['text'].values):
    vector = np.zeros(300)  # Initialize the vector for the current sentence
    tf_idf_weight = 0  # Initialize the total TF-IDF weight for the current sentence
    normalized_sentence = normalize(sentence)  # Normalize the sentence
    for word in normalized_sentence.split():  # Iterate through each word in the normalize
        if (word in w2v_model.wv.index_to_key) and (word in tfidf_words):  # Check if word
            vec = w2v_model.wv[word]  # Get the Word2Vec vector for the word
            tf_idf = dictionary[word] * (normalized_sentence.count(word) / len(normalized_
            vector += (vec * tf_idf)  # Add the TF-IDF weighted Word2Vec to the sentence v
            tf_idf_weight += tf_idf  # Accumulate the total TF-IDF weight
    if tf_idf_weight != 0:  # Check if any TF-IDF weighted vectors were added
        vector /= tf_idf_weight  # Compute the average TF-IDF weighted Word2Vec for the se
    tfidf_w2v_vectors.append(vector)  # Append the sentence vector to the list of TF-IDF w
```

```
100%|████████████████████████████████████████████████████████████████| 761
3/7613 [00:09<00:00, 775.05it/s]
```

In [138]:
```
X = np.array(tfidf_w2v_vectors)
y = train_clean2['target']
```

In [139]:
```
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42
```

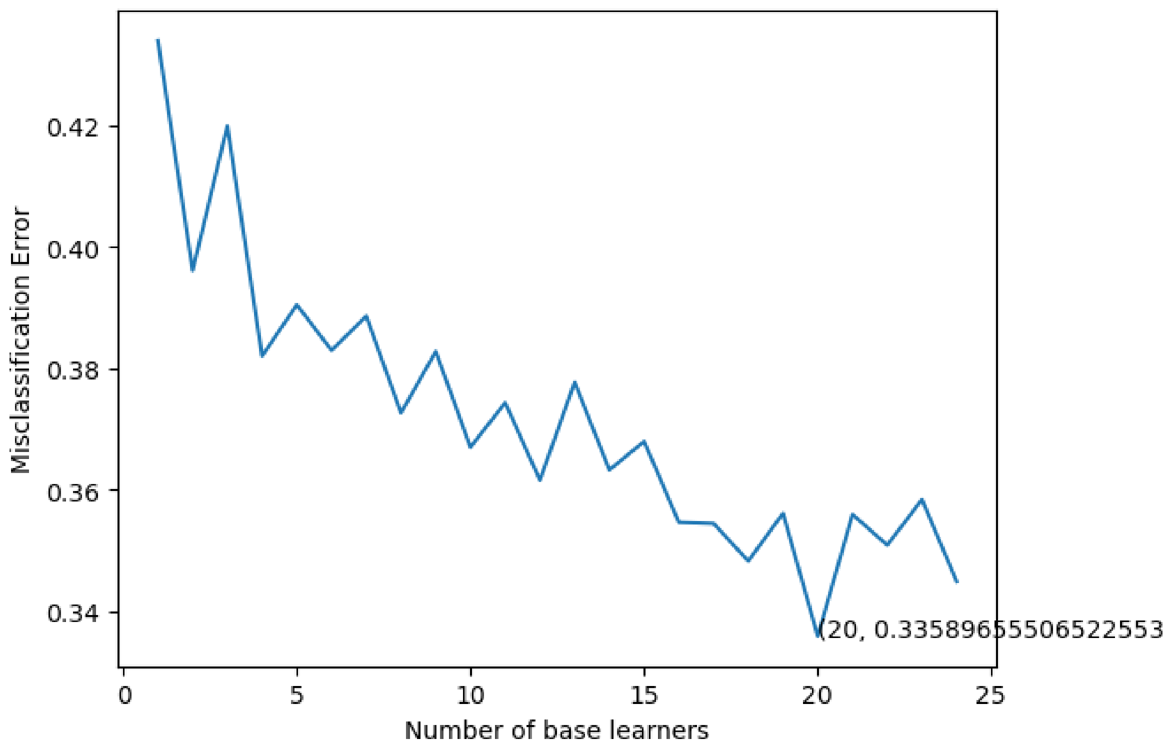
In [140]:
```
# Using Random Forest to build model for the classification of reviews.
forest = RandomForestClassifier(n_estimators=10, n_jobs=4)
forest.fit(X_train, y_train)
print(forest)
print(np.mean(cross_val_score(forest, X, y, cv=10)))
```

```
RandomForestClassifier(n_estimators=10, n_jobs=4)
0.5867636864051653
```

In [141]:
```
# Finding optimal number of base learners using k-fold CV ->
base_ln = [X for X in range(1, 25)]
```

In [142]:
```
# K-Fold Cross - validation
cv_scores = []
for b in base_ln:
    clf = RandomForestClassifier(n_estimators = b)
    scores = cross_val_score(clf, X_train, y_train, cv = 5, scoring = 'accuracy')
    cv_scores.append(scores.mean())
```

In [143]:
```
# Plot the misclassification error for each of estimators (Hint: Use the above code which
error = [1 - x for x in cv_scores] #error corresponds to e
optimal_learners = base_ln[error.index(min(error))] #Selection of optimal n
plt.plot(base_ln, error) #Plot between each nu o
xy = (optimal_learners, min(error))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.xlabel("Number of base learners")
plt.ylabel("Misclassification Error")
plt.show()
```



Observations:

- Misclassification error seems to be flattening out after 15 k-folds but there is still minor fluctuations.
- At 20 base learners, the misclassification error is about 33.5% - this is similar to model 3
- This is significantly higher than Models 1 and 2

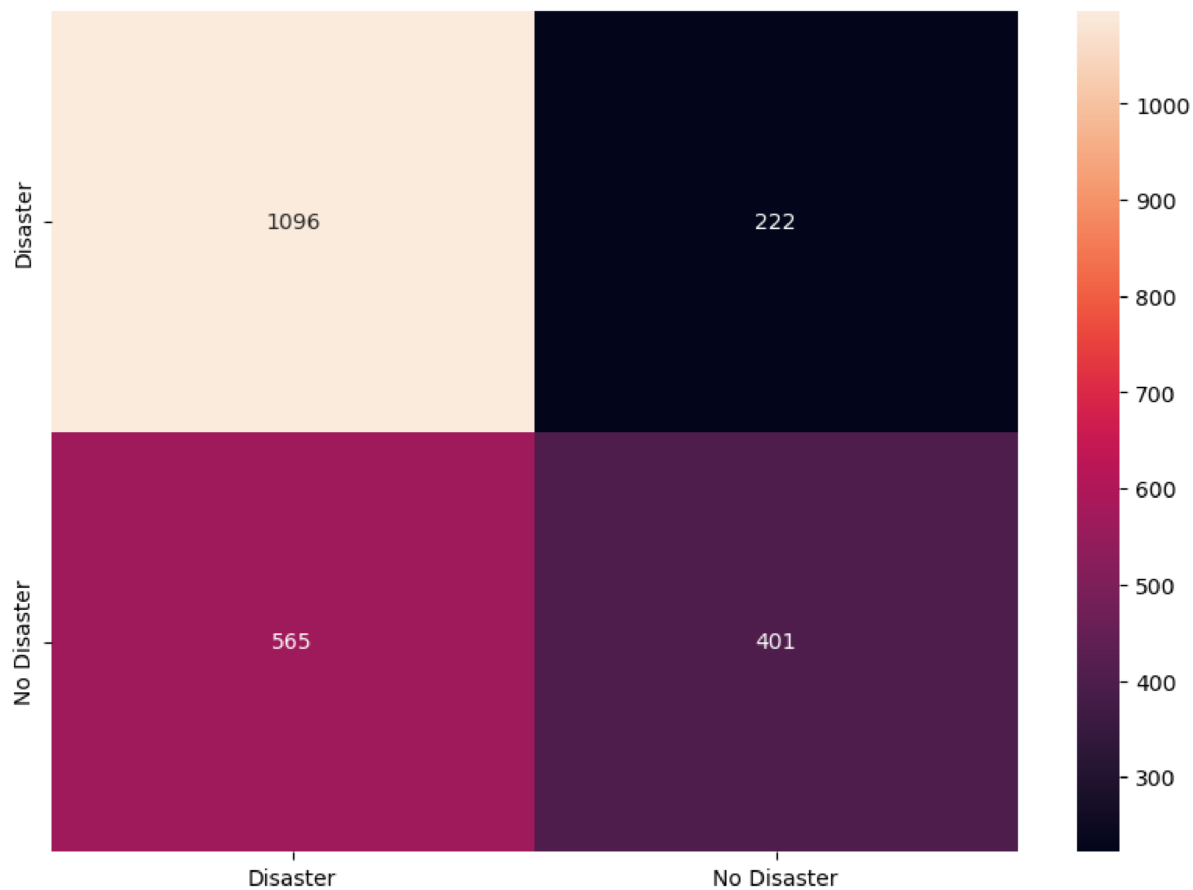
In [144]:
```
# Train the best model and calculating accuracy on test data
clf = RandomForestClassifier(n_estimators = optimal_learners)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

Out[144]: 0.6554290718038529

In [145]:
```
# Predict the result for test data using the model built above.
result = clf.predict(X_test)
```

In [146]:
```
# Plot the confusion matrix
conf_mat = confusion_matrix(y_test, result)

df_cm = pd.DataFrame(conf_mat, index = [i for i in ['Disaster', 'No Disaster']],
                     columns = [i for i in ['Disaster', 'No Disaster']])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

Out[146]: <AxesSubplot:>



Observations:

- 1096 Disaster tweets were classified correctly
- 401 No Disaster tweets were classified correctly

This model had is better at predicting Disasters and No Disasters accurately, but there are more False Negatives than Model 2. It appears that this model has successfully raised the accuracy score and the F-1 score, but at the expense of the recall scores.

In [147]:
```
#  Plotting the classification report
cr=metrics.classification_report(y_test, result) # From Project 2
print(cr)
```

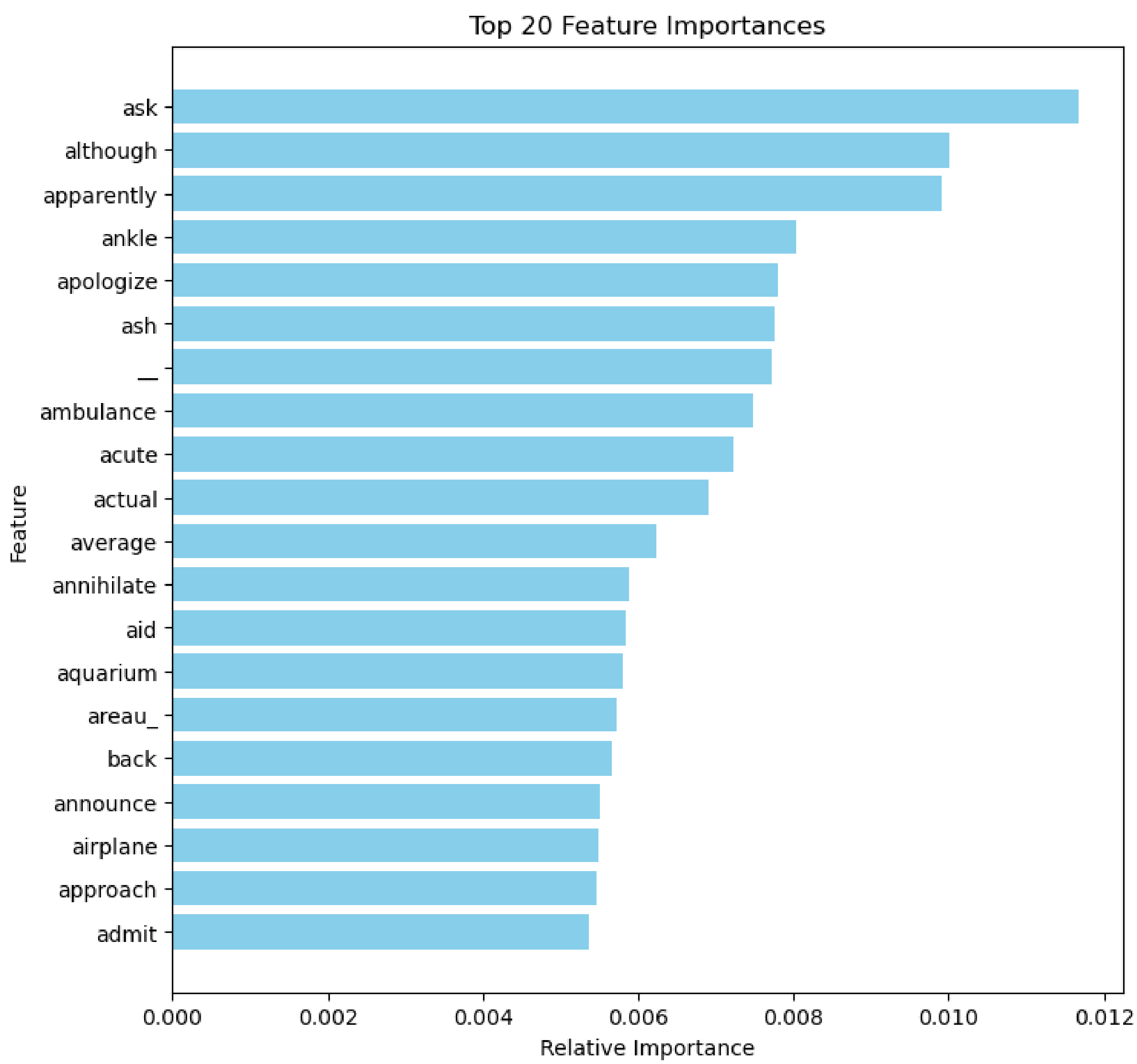
```
              precision    recall  f1-score   support

           0       0.66      0.83      0.74      1318
           1       0.64      0.42      0.50       966

    accuracy                           0.66      2284
   macro avg       0.65      0.62      0.62      2284
weighted avg       0.65      0.66      0.64      2284
```

Observations:

- Overall accuracy is about 66%
- False Negatives are higher than Model 1 or 2
- Accuracy and Recall scores are lower than our first two models

Feature Importance and Wordcloud of top 20 important features from Word2Vec+Randomforest based mode

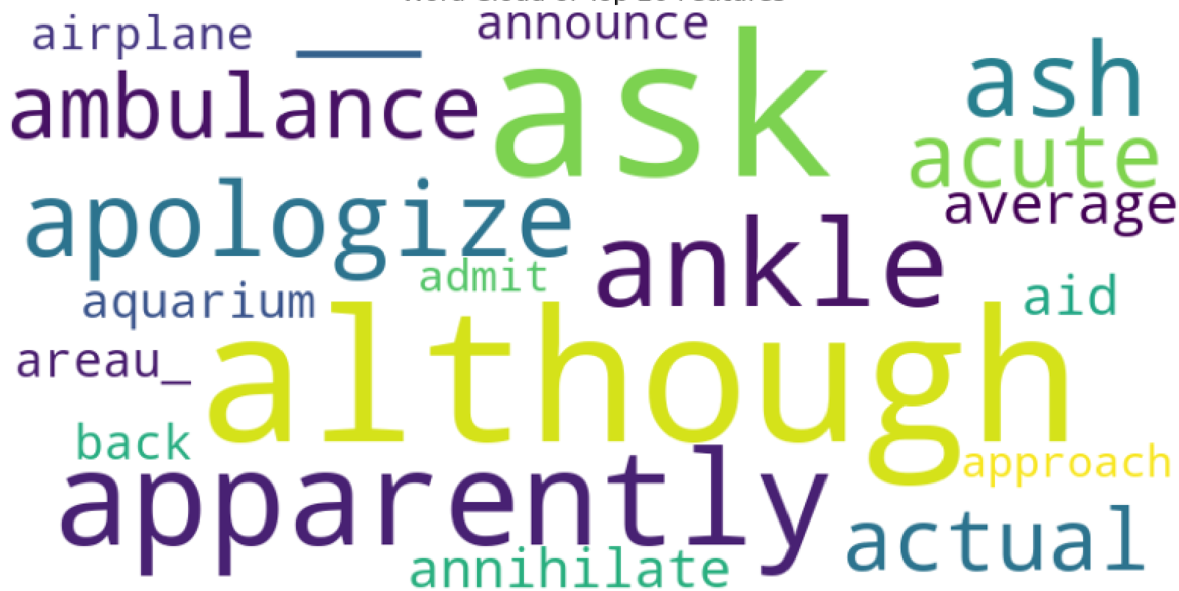
In [150]:
```
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]  # Sort indices in descending order

# Select only the top 20 features
top_indices = indices[:20]
top_importances = importances[top_indices][::-1]  # Reverse to plot in descending order
top_features = [all_features[i] for i in top_indices][::-1]  # Reverse to match importanc

plt.figure(figsize=(8, 8))
plt.title('Top 20 Feature Importances')
plt.barh(range(len(top_indices)), top_importances, color='skyblue', align='center')
plt.yticks(range(len(top_indices)), top_features)
plt.xlabel('Relative Importance')
plt.ylabel('Feature')
plt.show()
```



In [152]:

```
# Create a dictionary with words as keys and importances as values
word_importance_dict = {word: importance for word, importance in zip(top_features, top_im

# Generate the word cloud with importance-based sizing
wordcloud = WordCloud(width=800, height=400, background_color='white', prefer_horizontal=

# Plot the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud of Top 20 Features')
plt.axis("off")
plt.show()
```

Word Cloud of Top 20 Features



Conclusions for Model 4

- Overall accuracy of the model: 67%
- Recall scores dropped to 81% and 48%, respectively.
- The model does an inferior job classifying Disasters and False Negatives

This model did a much worse job than either of the previous two models

Overall Findings

Throughout our analysis of four sentiment analysis models, Model 2, utilizing TF-IDF, emerged as the most robust performer. With lower false negatives, superior accuracy, and commendable recall scores compared to the other models, Model 2 showcased its efficacy in effectively discerning sentiment from text data. Its utilization of TF-IDF for feature weighting enabled it to capture the importance of words within the context of the entire corpus, leading to more discriminative and accurate sentiment predictions. While Model 4, incorporating TF-IDF weighted Word2Vec, introduced an innovative fusion of techniques, its performance fluctuated and did not consistently outperform Model 2. Despite Model 4's attempt to capture semantic nuances through Word2Vec, its effectiveness was not conclusively superior to the TF-IDF approach of Model 2.

Best Model

Creating a .csv file that can be submitted to the the Kaggle competition.

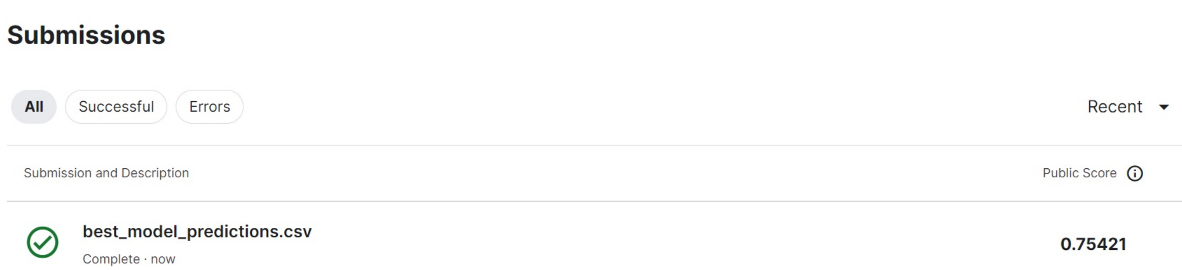
In [247]:
```python
import warnings
from sklearn.ensemble import RandomForestClassifier

# prediction
warnings.filterwarnings('ignore')

# Assuming clf is your trained RandomForestClassifier
clf = RandomForestClassifier(n_estimators=optimal_learners)
clf.fit(X_train, y_train)

# Make predictions on the test data
test_features = tfidf_vect.transform(test_clean['text'])
test_predictions = clf.predict(test_features)

# Create a DataFrame for predictions
pred_df = test_clean.drop(['text'], axis=1)
pred_df['target'] = test_predictions

# Save predictions to a CSV file
pred_df.to_csv('best_model_predictions.csv', index=False)

# Display the first few rows of the predictions DataFrame
pred_df.head()
```

Out[247]:

	id	target
0	0	1
1	2	1
2	3	1
3	9	0
4	11	1

In [248]:
```python
from IPython.display import Image

# Specify the path to your JPEG image file
image_path = 'score.jpg'

# Display the image
Image(filename=image_path)
```

Out[248]:



Final Score and Improving the model

I was able to achieve a score of 0.75421 out of 1.0 on my best model.

Improving Model 2, which is based on TF-IDF (Term Frequency-Inverse Document Frequency), involves several strategies aimed at enhancing its performance. Here are some suggestions to consider:

Feature Engineering:

Experiment with different text preprocessing techniques such as stemming, lemmatization, and handling of special characters. Explore different ways to handle stopwords, including removing them entirely or using custom stopword lists. Consider using n-grams (sequences of n words) in addition to unigrams to capture more contextual information.

Hyperparameter Tuning:

Perform grid search or randomized search to find the optimal hyperparameters for the Random Forest classifier, such as the number of estimators, maximum depth, and minimum samples per leaf. Experiment with different values for the max_features parameter in the TfidfVectorizer to control the number of features generated from the text data.

Ensemble Methods:

Explore ensemble methods like Gradient Boosting or XGBoost, which may offer improved performance over Random Forest. Consider using model stacking or blending techniques to combine predictions from multiple models for better accuracy. Advanced Text Representations:

Experiment with more sophisticated text embeddings like word embeddings (e.g., Word2Vec, GloVe) or contextual embeddings (e.g., BERT) to capture richer semantic information from the text.

Error Analysis:

Conduct thorough error analysis to identify common patterns of misclassification and areas for improvement. This can guide further iterations of feature engineering and model refinement.

Cross-Validation:

Ensure robustness of the model by performing cross-validation with multiple folds and assessing its performance on different subsets of the data.

By systematically exploring these strategies and iteratively refining the model based on experimentation and evaluation, you can incrementally enhance the performance of Model 2.

In [ ]:

```

```